ENGI 128
INTRODUCTION TO ENGINEERING SYSTEMS

Lecture 17:
Finite State Machines,
Behavior-Based Programming,
Final Design Challenge

“Understand Your Technical World”
Finite-State Machines
(Discrete Finite Automata)
Finite-State Machines (Discrete Finite Automata)

It sounds complicated, but you’ve been using them for a while:

Sample FSM diagram:

Q: What does this program do?
An Example FSM: Wall detection

[Demo dark avoid code on robot]
An Example FSM: Wall detection

reset

move forward
An Example FSM: Wall detection

reset → move forward → ? → ? → ? → reset
An Example FSM: Wall detection

- Reset
- Move forward
- Rotate left
- Rotate right

Questions:
- Move forward to rotate left
- Move forward to rotate right
- Move forward to question mark

States:
- Reset
- Move forward
- Rotate left
- Rotate right
Q: Note that the left sensor triggers a right rotate. Does this make sense?
An Example FSM: Wall detection

- **reset**
- **move forward**
  - left sensor detect wall
  - right sensor detect wall
  - no wall
- **rotate right**
- **rotate left**
An Example FSM: Wall detection

[Review dark avoid code in editor]
Finite-State Machines
(Quidditch Version)
For example

Let’s say you are programming a “Seeker” robot...
For example
Let’s say you are programming a “Seeker” robot...

reset

move around

wall detector

no event

avoid wall

wait for timer

timer done
For example

Let’s say you are programming a “Seeker” robot...

[Diagram of robot with states and transitions:]
- Start state: reset
- Move around
- Detect snitch
- Move to snitch
- Avoid wall
- Wall detector
- No event
- Timer done
- Wait for timer
- No snitch
- Move around
For example

Let’s say you are programming a “Seeker” robot...
Yikes!

Don’t worry, it looks simpler in code...
Behavior-Based Robot Programming
Behavior-Based Control

A behavior is a small program (or finite-state machine) that reads the sensors and controls the robot

- Each behavior only does **one simple** thing
- Each behavior has access to all the sensors of the robot and produces motor outputs (tv, rv, active)

Only one behavior can be active at a time

- There is a *prioritization* of behaviors
- More important ones override, or **subsume**, less important ones
An Example Behavior-Based Program

Follow a robot
- Sensor: IR Communications
- Behavior: Follow another robot

Avoid Obstacles
- Sensors: Bump sensor to detect wall
- Behavior: Move away from wall

Wander
- Sensor: Encoders
- Behavior: Move forward and turn

Q: Which behavior should have the highest priority?
Q: Which behavior should have the lowest priority?
Combining Behaviors

We combine behaviors by overriding, or *subsuming* lower-priority behaviors if a higher-priority behavior becomes active.

**Avoid Obstacle**
Behavior: avoid walls

**Follow a robot**
Behavior: Follow another robot

**Wander**
Behavior: move and turn

**Sensors**

**Actuators**

Highest priority

Subsume operator

Lowest priority
Genghis

The behavior-based poster child

Really simple hardware

- 6 legs, 2 motors per leg
  - $\alpha$-motor for forward/back, $\beta$-motor for up/down
- 2 bump sensors (feelers)
- 2 ground detection sensors (switches)
- 6 heat sensors (but they weren’t used for walking)

Genghis in Action
Subsumption Architecture

Combining Behaviors

We combine behaviors by overriding, or subsuming lower-priority behaviors if a higher-priority behavior becomes active.
Great, but how to you program this?

You can abstract this a bit more:

- Write each behavior as a function that returns a tuple of (tv, rv, active)
- write a `beh_subsume(high_priority, low_priority)` function that returns the high_priority or low_priority output, depending on which has active == True
- If neither behavior is active, it returns the INACTIVE_BEH output

```
INACTIVE_BEH = (False, 0.0, 0.0)
wander_out = wander(foo, bar)
gps_out = gps_navigation(bang, zoom)
obstacle_out = obstacle_avoidance(bif, bop)

beh = beh_subsume(gps_out, wander_out)
beh = beh_subsume(obstacle_out, beh)
velocity.set_tvr(beh_get_tv(beh), beh_get_rv(beh))
```
Great, but how to you program this?

There are several ways:

- Write each behavior as a function that returns a tuple of (active, tv, rv)
- Write getters to extract the different parts of the tuple
- Use if statements to overwrite the outputs of lower-priority behaviors

```python
INACTIVE_BEH = (False, 0.0, 0.0)
wander_out = wander(foo, bar)
gps_navigation_out = gps_navigation(bang, zoom)
obstacle_avoidance_out = obstacle_avoidance(bif, bop)

beh = INACTIVE_BEH
if beh_get_active(wander_out):
    beh = wander_out
if beh_get_active(gps_navigation_out):
    beh = gps_navigation_out
if beh_get_active(obstacle_avoidance_out):
    beh = obstacle_avoidance_out
velocity.set_tvrv(beh_get_tv(beh), beh_get_rv(beh))
```